Name - Gaurang A Raorane Div - D15A Roll no - 49 Batch - C

Experiment - 6

Aim: To study AJAX

Theory:

1. How do Synchronous and Asynchronous Requests differ?

Synchronous and asynchronous requests are two different ways of handling communication between a client (such as a web browser) and a server. Here's how they differ:

Synchronous Requests:

1. Blocking Behavior: In a synchronous request, the client sends a request to the server and waits for the server to process the request and send back a response. During this time, the client's execution is blocked, meaning it cannot perform any other tasks.

2. Sequential Execution: Synchronous requests are executed in sequence. The client must wait for each request to complete before sending the next one.

3. User Experience: Synchronous requests can sometimes lead to poor user experience, especially in cases where the server takes a long time to respond. The user may experience delays or unresponsiveness in the application.

4. Example: Traditional form submissions in web applications are synchronous. When a user submits a form, the browser sends the form data to the server and waits for a response before displaying the next page.

Asynchronous Requests:

1. Non-Blocking Behavior: In an asynchronous request, the client sends a request to the server but does not wait for the response immediately. Instead, it continues to execute other tasks while waiting for the response to arrive.

2. Parallel Execution: Asynchronous requests can be executed in parallel. The client can send multiple requests without waiting for each one to complete before sending the next one.

3. Improved Performance: Asynchronous requests can improve performance and responsiveness in applications, especially when dealing with long-running operations or multiple requests.

4. Example: AJAX (Asynchronous JavaScript and XML) requests in web applications are asynchronous. When a client makes an AJAX request to fetch data from the server, it continues executing other tasks while waiting for the response. Once the response is received, the client can process it without blocking the user interface.

In summary, synchronous requests block the execution of the client until a response is received, while asynchronous requests allow the client to continue executing other tasks while waiting for a response. Asynchronous requests are often preferred for web applications to improve performance and user experience.

1. Describe various properties and methods used in XMLHttpRequest Object

The XMLHttpRequest (XHR) object in JavaScript is used to interact with servers asynchronously. It enables web pages to send and receive data from a server without requiring a page reload. Here are some of the properties and methods commonly used with the XMLHttpRequest object:

Properties:

1. onreadystatechange: An event handler that is called whenever the `readyState` property changes. It typically points to a function that processes the server's response.

2. readyState: Represents the state of the XMLHttpRequest object. It can have values from 0 to 4, indicating the following states:

- 0: uninitialized

- 1: loading

- 2: loaded

- 3: interactive

- 4: complete

3. response: Represents the response body as a string, array buffer, blob, document, or JSON object, depending on the responseType property.

4. responseText: Returns the response as a string.

5. responseType: Specifies the type of data expected in the response. Possible values include "text", "arraybuffer", "blob", "document", or "json".

6. responseURL: Returns the serialized URL of the response or an empty string if the URL is null.

7. status: Returns the HTTP status code of the response (e.g., 200 for a successful request).

8. statusText: Returns the HTTP status text of the response (e.g., "OK" for a successful request).

9. timeout: Represents the number of milliseconds a request can take before automatically being terminated.

Methods:

1. abort(): Cancels the current request.

2. open(method, url[, async[, user[, password]]]): Initializes a request. It specifies the method (e.g., "GET", "POST"), URL, and whether the request should be asynchronous. Optional parameters include the username and password for authentication.

3. send([body]): Sends the request to the server. The optional body parameter can contain the request payload for methods like "POST" or "PUT".

4. setRequestHeader(name, value): Sets the value of an HTTP request header.

5. getAllResponseHeaders(): Returns all the response headers as a string.

6. getResponseHeader(name): Returns the value of a specific response header.

These properties and methods allow developers to initiate HTTP requests, monitor the progress of requests, handle responses, and manipulate request and response headers. By leveraging the XMLHttpRequest object, developers can create dynamic and interactive web applications that communicate with servers asynchronously.

Code -

app.py -

from flask import Flask, render\_template, request, jsonify

import json

app = Flask(\_\_name\_\_)

# Load existing registrations from JSON file

try:

with open('registrations.json', 'r') as file:

registrations = json.load(file)

except (FileNotFoundError, json.JSONDecodeError):

registrations = []

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/register', methods=['POST'])

def register():

data = request.json

# Validate the form data

name = data.get('name')

college = data.get('college')

username = data.get('username')

password = data.get('password')

confirm\_password = data.get('confirmPassword')

if (name and college and username and password and confirm\_password):

return jsonify({'error': 'All fields are required'}), 400

if password != confirm\_password:

return jsonify({'error': 'Passwords do not match'}), 400

for reg in registrations:

if reg['username'] == username:

return jsonify({'error': 'Username already exists'}), 400

# Create a new registration dictionary

new\_registration = {

'name': name,

'college': college,

'username': username,

'password': password

}

# Add the new registration to the registrations list

registrations.append(new\_registration)

# Save the registrations to a JSON file

with open('registrations.json', 'w') as file:

json.dump(registrations, file)

return jsonify({'message': 'Successfully registered'}), 200

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Index.html -

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Registration Page</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f5f5f5;

margin: 0;

padding: 0;

}

h2 {

text-align: center;

margin-top: 20px;

}

form {

max-width: 400px;

margin: 0 auto;

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

form div {

margin-bottom: 15px;

}

label {

display: block;

font-weight: bold;

margin-bottom: 5px;

}

input[type="text"],

input[type="password"] {

width: calc(100% - 10px);

padding: 8px;

border: 1px solid #ccc;

border-radius: 5px;

font-size: 16px;

}

input[type="submit"] {

width: 100%;

background-color: #4caf50;

color: #fff;

border: none;

padding: 10px;

border-radius: 5px;

cursor: pointer;

font-size: 16px;

}

input[type="submit"]:hover {

background-color: #45a049;

}

#message {

display: none;

color: green;

margin-top: 10px;

}

</style>

</head>

<body>

<h2>Registration Form</h2>

<form id="registrationForm" onsubmit="submitForm(event)">

<div>

<label for="name">Name:</label>

<input type="text" id="name" required>

</div>

<div>

<label for="college">College:</label>

<input type="text" id="college" list="colleges" required>

<datalist id="colleges">

<!-- Autocomplete options will be filled dynamically -->

</datalist>

</div>

<div>

<label for="username">Username:</label>

<input type="text" id="username" required>

</div>

<div>

<label for="password">Password:</label>

<input type="password" id="password" required>

</div>

<div>

<label for="confirmPassword">Confirm Password:</label>

<input type="password" id="confirmPassword" required>

</div>

<button type="submit">Register</button>

</form>

<div id="message">Successfully Registered</div>

<script>

// Simulated autocomplete for college names

var colleges = ['College A', 'College B', 'College C', 'College D'];

var collegeInput = document.getElementById('college');

var collegeList = document.getElementById('colleges');

colleges.forEach(function(college) {

var option = document.createElement('option');

option.value = college;

collegeList.appendChild(option);

});

collegeInput.setAttribute('list', 'colleges');

function submitForm(event) {

event.preventDefault();

var name = document.getElementById('name').value;

var college = document.getElementById('college').value;

var username = document.getElementById('username').value;

var password = document.getElementById('password').value;

var confirmPassword = document.getElementById('confirmPassword').value;

if (name === '') {

alert('Name field cannot be empty');

return;

}

if (password !== confirmPassword) {

alert('Passwords do not match');

return;

}

var formData = {

name: name,

college: college,

username: username,

password: password

};

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() {

if (xhr.readyState === XMLHttpRequest.DONE) {

if (xhr.status === 200) {

document.getElementById('message').style.display = 'block';

} else {

alert('Error: ' + xhr.responseText);

}

}

};

xhr.open('POST', '/register', true);

xhr.setRequestHeader('Content-Type', 'application/json');

xhr.send(JSON.stringify(formData));

}

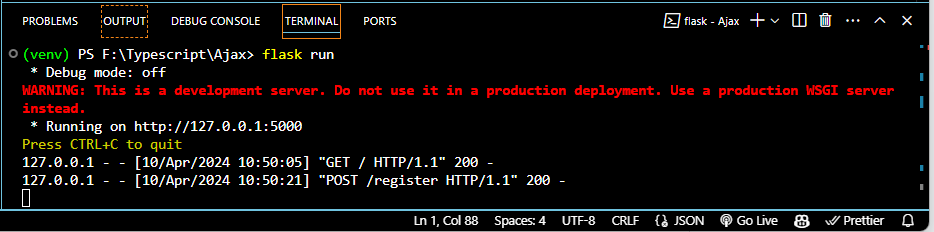
</script>

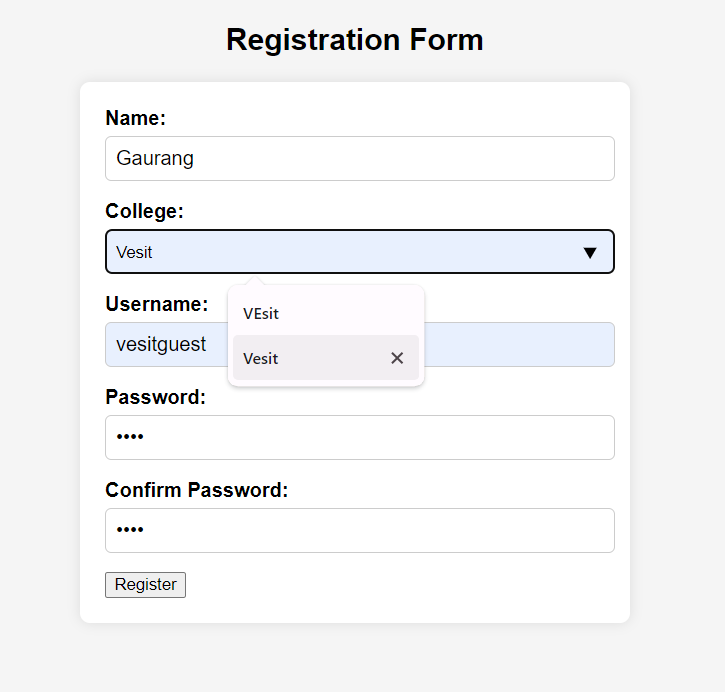
</body>

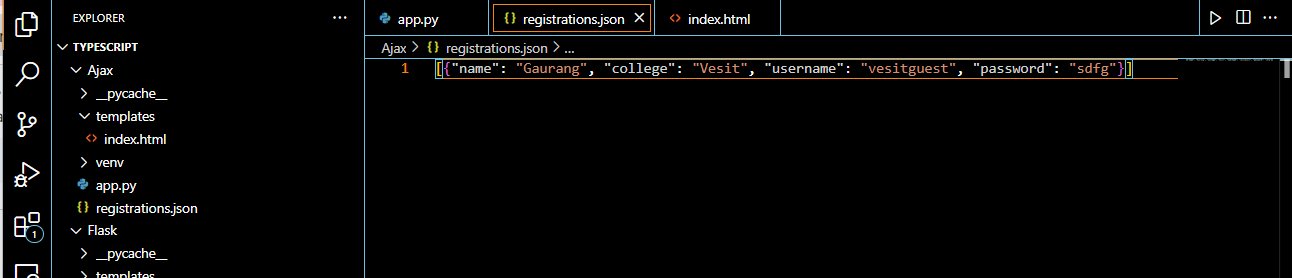
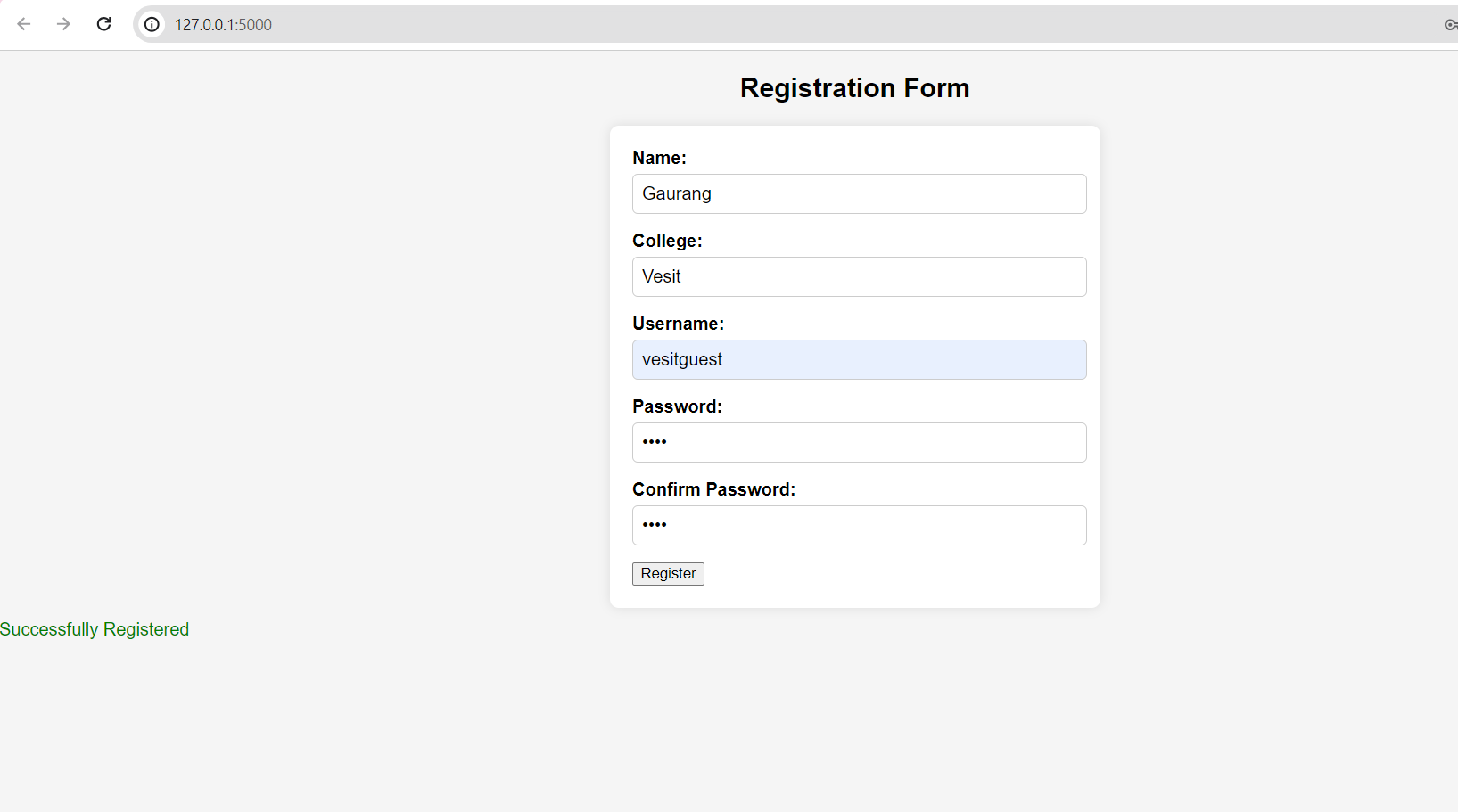
</html>

Registration.json -

[{"name": "Gaurang", "college": "Vesit", "username": "vesitguest", "password": "sdfg"}]







Conclusion -

Through this experiment we understood AJAX and implemented it successfully in our code. We understood asynchronous and synchronous requests and used the XMLhttprequest object to serve asynchronous requests.